Tests API (Vitest + Supertest)

Objectif

Mettre en place des tests automatisés reproductibles pour une API Node.js (Fastify), afin de :

- valider les endpoints HTTP
- éviter les régressions
- tester sans interface graphique (Next.js pas nécessaire)

Installation

Vitest : le moteur de test

Installer les dépendances :

```
npm install -D vitest supertest
npm install -D @types/supertest
```

Structure du projet

Exemple :

```
/src
  app.ts
  modules/
/tests
  user.test.ts
```

1. Rendre l'application testable

Dans `app.ts` :

```
import Fastify from "fastify"

export const buildApp = () => {
  const app = Fastify()
  app.get("/health", async () => {
    return { status: "ok" }
  })
  return app
}
```

 Important :

ne pas faire `app.listen()` ici on exporte une fonction pour créer une instance propre à chaque test

Mettre le lancement du serveur dans un fichier à part :

```
import {buildApp} from "./app";

const app = buildApp()

app.listen({ port: 3000 }).then(() => {
    console.log("Server running on http://localhost:3000")
})
```

2. Premier test

Créer `/tests/health.test.ts` :

```
let app:FastifyInstance

beforeAll(async () => {
    app = buildApp()
    await app.ready()
})

afterAll(async () => {
    await app.close()
})
describe("GET /health", () => {
    it("should return status ok", async () => {
        const res = await request(app.server)
            .get("/health")

        expect(res.status).toBe(200)
        expect(res.body.status).toBe("ok")
    })
})
```

3. Script de test

Dans `package.json` :

```
{
  "scripts": {
    "test": "vitest"
  }
}
```

Lancer les tests :

```
npm run test
```

4. Tester un vrai endpoint (ex: User)

Exemple :

```
describe("POST /users", () => {
  it("should create a user", async () => {
    const app = buildApp()

const res = await request(app.server)
  .post("/users")
  .send({
    email: "test@test.com",
    password: "123456"
  })

expect(res.status).toBe(201)
expect(res.body.data.email).toBe("test@test.com")
  })
})
```

5. Base de données de test

Point critique : les tests doivent être isolés

Solutions :

- SQLite en mémoire
- ou base dédiée “test”

Exemple simple avec reset :

```
beforeEach(async () => {
  await prisma.user.deleteMany()
})
```

6. Bonnes pratiques

- 1 test = 1 comportement
- ne pas dépendre d’un autre test
- utiliser des données simples
- vérifier :
  - status HTTP
  - structure de réponse
  - données retournées

7. Erreurs à éviter

- Tester uniquement les services→ on veut tester l'API HTTP complète
- Utiliser des mocks partout→ tests irréalistes
- Ne pas nettoyer la base→ tests instables (flaky)

Résultat attendu

À la fin :

- les endpoints sont testés automatiquement
- les tests sont relançables à l'infini

Bonus

Commandes utiles :

```
npm run test -- --watch
```

→ relance automatique des tests